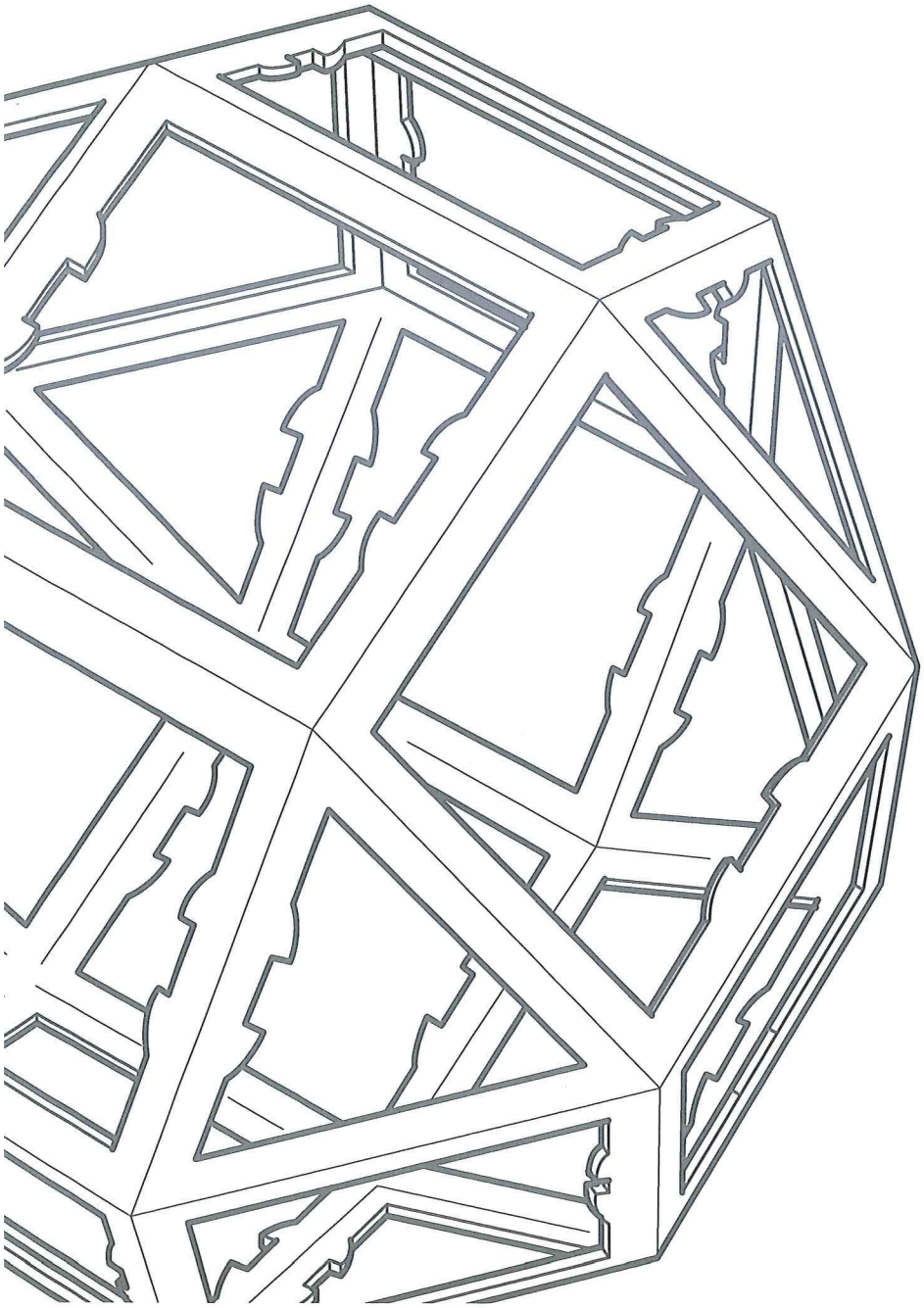
Department of Mathematics and Statistics
College of Engineering

UC
UNIVERSITY OF CANTERBURY
Te Whare Wānanga o Waitaha
CHRISTCHURCH NEW ZEALAND

Summer Research Project

Teaching Computers to See

by Brendan Bycroft



09

Teaching Computers to See

Brendan Bycroft
Supervisor: Dr Richard Brown
University of Canterbury Summer Project

February 5, 2009

Abstract

A program was created which enabled 3D reconstruction of a scene given two images from a simple camera. This was achieved using a combination of linear algebra and homogeneous coordinates. The reconstruction was separated into the calibration of the cameras, followed by determination of 3D positions using point correspondences.

1 Introduction

1.1 Overview

Computer vision is the task of extracting useful information from an image, or series of images using a computer. This can be used in applications such as artificial intelligence. One of the problems in computer vision is calculating where objects are in 3D space. Images from a camera are inherently 2 dimensional, and thus do not provide direct information about distances and shapes of 3 dimensional objects.

Humans are able to easily discern positions, sizes and shapes of objects from a single image. However, given the large amount of a priori knowledge required, it is not possible to easily get a computer to do this.

Instead, a number of images can be taken of a particular object or a scene. If a series of points of interest can be matched between the images, the process of reconstructing the scene is simplified somewhat. Such points of interest typically mean sharp corners and edges, which can be tracked with relative ease.

Two images of a scene hold almost enough information for an accurate reproduction of the scene in 3D. An extra piece of information is required about the cameras used. For a simple camera, this remains constant no matter how the camera is positioned. Once this is known, the scene can be fully determined relative to the cameras except for knowledge of scale.

The methods used here uses homogeneous coordinates extensively, so that a number of transformations, such as the camera projection, can be done using linear algebra.

From the points known to correspond, the relationship between the two cameras can be determined. Using that information, the 3D location of the points found in the images can be calculated.

This 3D reconstruction process was implemented in the Python programming language by the author.

In Section 1, homogeneous coordinates and the camera model are introduced. In Section 2, the mathematical basis for 3D reconstruction is explained. Section 3 then describes the implementation of the algorithms using the mathematics from Section 2 and also describes attempts at using various image processing algorithms.

1.2 Homogeneous Coordinates

Homogeneous coordinates are used extensively in the scene reconstruction. As such, some basic constructs will be defined here.

In 2D Cartesian coordinates, a point is represented by $(x, y)^T$. Its equivalent in homogeneous coordinates is $(x, y, 1)^T$. However, this is the same as $(kx, ky, k)^T$, where $k \in \mathbb{R}$ is a constant. Therefore, for each point in a Cartesian plane there is a set of homogeneous coordinates. If a homogeneous coordinate is given by $(x, y, w)^T$, then its Cartesian coordinate is given by $(x/w, y/w)^T$.

A line in 2D has the form $ax + by + c = 0$. In the homogeneous coordinate system, this can be represented as $\mathbf{l} = (a, b, c)^T$. A homogeneous point, $\mathbf{a}$ lies on a line $\mathbf{l}$ if

$$\mathbf{l} \cdot \mathbf{a} = 0 \tag{1}$$

Furthermore, a line which passes through two points can be found by

$$\mathbf{l} = \mathbf{a} \times \mathbf{a}' \tag{2}$$

That is, the cross product of the two points. Also, the intersection of two lines is

$$\mathbf{a} = \mathbf{l} \times \mathbf{l}' \tag{3}$$

In the case where a homogeneous coordinate has its last value equal to 0, its Cartesian coordinate is not defined. However, in 2D homogeneous coordinates, that point represents a point on the line at infinity, $\mathbf{l}_\infty = (0, 0, 1)$, and is called an ideal point. In 3D, the point will lie on the plane at infinity, $\pi_\infty = (0, 0, 0, 1)$.

The set of homogeneous coordinates in 2D made up of all ideal and non-ideal points form the projective plane $\mathbb{P}^2$[4, p26].

Another important construct is the conic. A conic can be represented in homogeneous coordinates by the equation

$$ax^2 + by^2 + 2cxy + 2dxw + 2eyw + fw^2 = 0 \tag{4}$$

This can be conveniently represented in matrix notation as

$$\begin{bmatrix} x & y & w \end{bmatrix} \begin{bmatrix} a & c & d \\ c & b & e \\ d & e & f \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = 0 \tag{5}$$

$$\mathbf{x}^T C \mathbf{x} = 0 \tag{6}$$

A homography, also known as a projective transform, is a mapping of points from one plane to another. That is, it maps $\mathbb{P}^2 \mapsto \mathbb{P}^2$. A homography is represented by a 3×3 invertible matrix and preserves straight lines[4, p32].

1.3 The Camera Model

An accurate model to describe real-world cameras is the pin-hole camera model. A pinhole camera is constructed with a small hole on one screen in front of another screen as shown in Figure 1. Rays of light from points in the scene pass through the small hole and land on the screen behind. The hole means that a particular light source will only reach a single point on the screen. In common, simple cameras, such as those found in a cell-phone or web-cam, the pinhole is replaced with a convex lens, which has the same effect.

Mathematically, the pinhole is defined as the camera centre, $\mathbf{C}$, and the focal length, f, is the distance from the pinhole to the other screen. In this model, the screen on which the image is projected can be visualized as being in front of the camera centre without loss of generality.

Points in 3-space are represented by homogeneous coordinates $\mathbf{X} = (x, y, z, w)^T$. The effect of a camera is to project a point $\mathbf{X}$ onto the 2D screen. This screen is also represented by homogeneous coordinates, *i.e.* the projective space $\mathbb{P}^2$. This means that the properties of a camera can be expressed by a 3×4 projective matrix P, called the camera matrix.

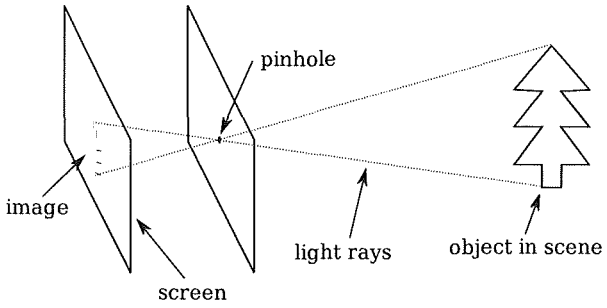


Figure 1: A pinhole camera.

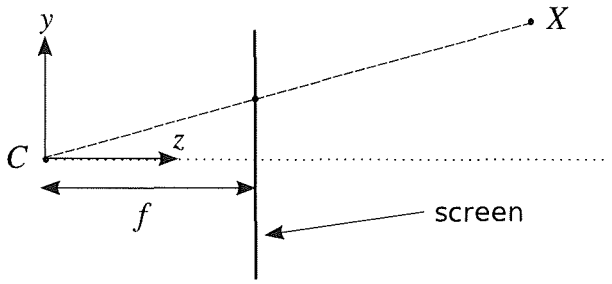


Figure 2: Cross-section of the pin-hole camera model. $\mathbf{C}$ is the camera centre located at the origin. The focal length, f, is the distance from the camera centre to the screen. A point in 3-space, $\mathbf{X}$, is projected onto the screen.

This camera matrix can be decomposed into more useful matrices, which describe its position, orientation and internal camera properties. The decomposition is

$$P = KR[I|-\mathbf{C}] \tag{7}$$

where

$$K = \begin{bmatrix} \alpha_x & s & p_x \\ 0 & \alpha_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

represents the camera's internal parameters, R is a 3×3 rotation matrix representing the camera's orientation, I is the identity, and $\mathbf{C}$ is the camera centre.

These results come about as follows:

A camera is positioned with its centre at the origin, and pointing up along the z-axis. This frame of reference is the camera frame with homogeneous coordinates $\mathbf{X}_{cam}$. The screen is represented by the plane $z = f$. A point at position $\mathbf{X}_{cam} = (X, Y, Z, 1)^T$ is projected onto the screen as shown in Figure 2. This means that it will be imaged on the screen at the point $\mathbf{x} = (fX/Z + p_x, fY/Z + p_y)^T$. The point $(p_x, p_y)^T$ is the offset of the screen origin from the z-axis. Expressed in homogeneous coordinates this becomes $\mathbf{x} = (fX + Zp_x, fY + Zp_y, Z)^T$. Note that the use of homogeneous coordinates removes the need for division so that the transformation can be linear. In matrix form,

$$\begin{bmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{9}$$

$$\mathbf{x} = \hat{P}\mathbf{X}_{cam} \tag{10}$$

Thus, for a camera with a fixed position at the origin and facing vertically upwards, the camera model is represented by the matrix:

$$\hat{P} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{11}$$

The left-hand 3×3 sub-matrix is K, the calibration matrix, so that

$$\hat{P} = [K|0] = K[I|0] \tag{12}$$

A camera can also have any position and orientation within a scene. This is modelled by transforming the scene coordinates, $\mathbf{X}$, to the camera coordinates of above. This can be done by the transformation

$$\mathbf{X}_{cam} = R(\mathbf{X} - \mathbf{C}) = R\mathbf{X} - R\mathbf{C} \tag{13}$$

where $\mathbf{X}$ is a point in the scene, and $\mathbf{X}_{cam}$ is the corresponding point in the camera coordinate system. R is a 3×3 rotation matrix which represents the orientation of the camera, and $\mathbf{C}$ is the camera centre. This equation can be written in homogeneous coordinates:

$$\mathbf{X}_{cam} = \begin{bmatrix} R & -R\mathbf{C} \\ 0 & 1 \end{bmatrix} \mathbf{X} \tag{14}$$

Now, since $\mathbf{x} = \hat{P}\mathbf{X}_{cam}$, and from Equations 12 & 14, we get

$$\mathbf{x} = K[I|0] \begin{bmatrix} R & -R\mathbf{C} \\ 0 & 1 \end{bmatrix} \mathbf{X} = K[R| - R\mathbf{C}]\mathbf{X} = KR[I| - \mathbf{C}]\mathbf{X} \tag{15}$$

where

$$P = KR[I| - \mathbf{C}] \tag{16}$$

is the projection matrix.

2 3D Scene Reconstruction

2.1 Calculation of a Camera's Internal Parameters

The calculation of the calibration matrix K of a camera is a crucial step in being able to reconstruct a scene correctly. This is because the camera matrices can only be calculated up to a projective transform from points that are known to correspond.

There are a number of ways to find a camera's calibration matrix. They typically involve using information such as where two lines are meant to be parallel or perpendicular. In the reference implementation, a different method was used which involves square planes.

In this method, three squares are placed in a scene, such that they are at an angle to each other. The squares are imaged with the camera to be calibrated. The corners of each of the squares are then identified in the image. Knowing these points, the homography from a unit square to a square in the image is calculated. That is, the 3×3 matrix, H, which transforms the points from the unit square to the square in the image in homogeneous coordinates.

The shape used (a square) is arbitrary. However, it is necessary to provide at least 4 coplanar points which represent a known shape, and hence a square is the simplest.

From a set of three homographies, a matrix representing the image of the absolute conic (IAC) can be calculated.

The absolute conic is a conic which lies on the plane at infinity, π_∞. Points on this plane can be represented by $\mathbf{X}_\infty = (\mathbf{d}^T, 0)^T$. As imaged by a camera $P = KR[I| - \mathbf{C}]$, these points become

$$\mathbf{x} = P\mathbf{X}_\infty = KR[I| - \mathbf{C}] \begin{pmatrix} \mathbf{d} \\ 0 \end{pmatrix} = KR\mathbf{d} \tag{17}$$

As this is a mapping of points from one plane to another, it can be described as a homography, $\mathbf{x} = H\mathbf{d}$, where $H = KR$.

A conic, C, under a homography, H, will map as $C \mapsto H^{-T}CH^{-1}$. If $H = KR$ and the conic is the absolute conic, with $C = \Omega_\infty = I_{3\times3}$, then the mapping becomes

$$\begin{aligned}\omega &= (KR)^{-T}I(KR)^{-1} \\ \omega &= K^{-T}RR^{-1}K^{-1} \\ \omega &= (KK^T)^{-1}\end{aligned} \tag{18}$$

K is an upper triangular matrix. This means it can be determined uniquely from ω by the Cholesky factorization of $KK^T = \omega^{-1}$[7].

In $\mathbb{P}^3$, planes have the property that they intersect the absolute conic Ω_∞ at the 'circular points', which are $(1, \pm i, 0)$??p52]hartley2003 and the imaged circular points are $H(1, \pm i, 0)$. If $H = (\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3)$, then the imaged circular points become $\mathbf{h}_1 \pm i\mathbf{h}_2$. These two points lie on the IAC, so together they provide two constraints on the position of ω. These constraints can be written as $(\mathbf{h}_1 \pm i\mathbf{h}_2)\omega(\mathbf{h}_1 \pm i\mathbf{h}_2) = 0$. This can be better represented with all real numbers by separating the real and imaginary parts:

$$\mathbf{h}_1^T\omega\mathbf{h}_2 = 0 \text{ and } \mathbf{h}_1^T\omega\mathbf{h}_1 = \mathbf{h}_2^T\omega\mathbf{h}_2 \tag{19}$$

For a conic to be fully determined, 5 points on it must be known. Here, each homography provides two such points, and so three homographies are needed to calculate the IAC.

The constraints of Equations 19 are linear in ω. This means that it is possible to construct a system $A\mathbf{w} = \mathbf{0}$ where $\mathbf{w}$ is ω in the form of a 6-vector. That is,

$$\omega = \begin{bmatrix} w_1 & w_2 & w_4 \\ w_2 & w_3 & w_5 \\ w_4 & w_5 & w_6 \end{bmatrix} \tag{20}$$

where w_i is the ith component of $\mathbf{w}$.

The system $A\mathbf{w} = \mathbf{0}$ can be solved using the singular value decomposition (SVD)[8], where A is decomposed into USV^H. The vector $\mathbf{w}$ is then the column of V which corresponds to the smallest value in the diagonal of S (In computer implementations, this is usually the last column).

With ω known, the calibration matrix, K, can be calculated using the Cholesky factorization $\omega^{-1} = KK^T$.

2.2 The Fundamental Matrix

The fundamental matrix is a rank 2, 3×3 matrix which defines the relationship between two cameras P and P' which are imaging the same scene. The fundamental matrix is useful because it can be found directly from a set of point correspondences in the two images, and subsequently used to calculate the camera matrices.

As shown in Figure 3, an imaged point on one screen could be the image of any 3D point on the ray back-projected through it from the camera centre. This means that the possible corresponding image points on a second screen must be on a line. The connection between the point on one screen and its corresponding line on the other is given by the relation

$$\mathbf{l}' = F\mathbf{x} \tag{21}$$

where $\mathbf{x}$ is a point on P, $\mathbf{l}'$ is its corresponding line on P', and F is the fundamental matrix.

The line $\mathbf{l}'$ is called an epipolar line. A notable point the screen of P' is called the epipole. At this point, all possible epipolar lines meet. It is also the projection of the first camera's centre, $\mathbf{C}$, onto the screen of P. It is known that if a point is imaged at $\mathbf{x}$ on the first camera, then that same point imaged on the second camera must be at a point $\mathbf{x}'$ which is on the epipolar line given by $\mathbf{l}' = F\mathbf{x}$. This, and Equation 1 give the constraint on F that

$$\begin{aligned}\mathbf{x}'^T\mathbf{l}' &= 0 \\ \mathbf{x}'^TF\mathbf{x} &= 0\end{aligned} \tag{22}$$

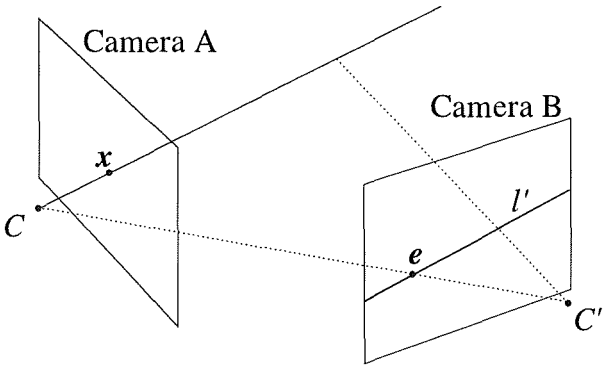


Figure 3: $\mathbf{C}$ and $\mathbf{C}'$ are the camera centres of cameras A and B respectively. The line through $\mathbf{C}$ and $\mathbf{x}$ is the back-projected ray of $\mathbf{x}$, and $\mathbf{l}'$ is that ray's image on camera B. A point imaged at $\mathbf{x}$ on camera A must be imaged on the line $\mathbf{l}'$ for camera B. The point $\mathbf{e}$ is the epipole corresponding to camera A.

which is linear in F. This means that if a set of point correspondences $\mathbf{x} \leftrightarrow \mathbf{x}'$ between two images are known, the fundamental matrix F can be found. That is, the constraints can be written in the form

$$A\mathbf{f} = 0 \tag{23}$$

where $\mathbf{f}$ is a 9 vector with the values of the fundamental matrix F in row-major order, and A is a $n \times 9$ matrix. Using Equation 22, a row of A can be generated using two corresponding points. At least 8 rows of A are required for $\mathbf{f}$ to be uniquely determined, where each row comes from one set of corresponding points.

From Equation 23, it is clear that $\mathbf{f}$ is the right null-space of A. This is best calculated using the singular value decomposition with $A = USV^H$ to find the least squares solution, where the vector $\mathbf{f}$ is the last column of V.

2.3 Calculation of Camera Matrices

Once both the fundamental matrix, F, and the calibration matrix, K, are known, the camera matrices, P and P' can also be found. The imaged points are first converted to normalized coordinates. This conversion is given by the equation

$$\hat{\mathbf{x}} = K^{-1}\mathbf{x} \tag{24}$$

where $\mathbf{x}$ is a point on the image, and $\hat{\mathbf{x}}$ is the corresponding normalized point. This means that given a general camera matrix, $P = KR[I \mid -\mathbf{C}]$,

$$\hat{\mathbf{x}} = [R \mid -\mathbf{C}]\mathbf{X} \tag{25}$$

where $\mathbf{X}$ is in world coordinates.

The normalization leads to a matrix similar in function to the fundamental matrix called the essential matrix. The essential matrix has fewer degrees of freedom, and has the property that the camera matrices can be easily computed from it. The calculation of the essential matrix is as follows:

Similar to Equation 22 of the fundamental matrix, we define the essential matrix as having the property

$$\hat{\mathbf{x}}'^T E\hat{\mathbf{x}} = 0 \tag{26}$$

where $\hat{\mathbf{x}}'$ and $\hat{\mathbf{x}}$ have been normalized and E is the essential matrix. Substituting in Equation 24, this becomes

$$\mathbf{x}'^T K^{-T} E K^{-1}\mathbf{x} = 0 \tag{27}$$

so that

$$K^{-T}EK^{-1} = F$$
$$E = K'^{T}FK \tag{28}$$

Now, as the essential matrix can be calculated, it is used to find the camera matrices. At this point, it is not known where two cameras are in the world coordinate frame. As such, the first camera is arbitrarily placed at the origin, so that

$$P = [I|0] \tag{29}$$

Here, the camera is defined in normalized coordinates. The other camera, P', is then one of four choices. These are

$$P' = [UWV^T| + \mathbf{u}_3] \text{ or } [UWV^T| - \mathbf{u}_3] \text{ or } [UW^TV^T| + \mathbf{u}_3] \text{ or } [UW^TV^T| - \mathbf{u}_3] \tag{30}$$

where $E = USV^T$ as calculated by the SVD, $\mathbf{u}_3$ is the last column of U, and

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{31}$$

See [4, p258] for the derivation. The correct choice is known as it will be the only one which calculates the 3D points as being in front of both cameras.

2.4 Computation of points in 3D space

When the camera matrices P and P' of the two cameras are calculated, it is then possible to calculate the location of the original points in 3D. We know that $\mathbf{x} = P\mathbf{X}$ and $\mathbf{x}' = P'\mathbf{X}$, where the only unknown is $\mathbf{X}$.

Geometrically, finding the point $\mathbf{X}$ is equivalent to finding the intersection of the back-projected lines formed from $\mathbf{x}$, $\mathbf{x}'$ and their respective cameras as shown in Figure 4. However, due to numerical inaccuracies in the camera matrices and the imaged point locations, these two lines will be skew.

This is overcome by using the singular value decomposition to calculate the most likely point. This is done by creating the system $A\mathbf{X} = 0$ from $\mathbf{x} = P\mathbf{X}$ and $\mathbf{x}' = P'\mathbf{X}$.

$$\mathbf{x} = P\mathbf{X} \tag{32}$$
$$\mathbf{x} \times \mathbf{x} = \mathbf{x} \times (P\mathbf{X}) \tag{33}$$
$$0 = \mathbf{x} \times (P\mathbf{X}) \tag{34}$$

Expanding this gives three equations linear in $\mathbf{X}$ of which two are linearly independent. Therefore, using equations from both cameras, four equations can be found linear in $\mathbf{X}$. This is enough to solve the equation $A\mathbf{X} = 0$ for $\mathbf{X}$ where

$$A = \begin{bmatrix} x\mathbf{p}^{3T} - \mathbf{p}^{1T} \\ y\mathbf{p}^{3T} - \mathbf{p}^{2T} \\ x'\mathbf{p}'^{3T} - \mathbf{p}'^{1T} \\ y'\mathbf{p}'^{3T} - \mathbf{p}'^{2T} \end{bmatrix} \tag{35}$$

and $\mathbf{x} = (x, y)$ and $\mathbf{p}^{iT}$ is the ith row of P[4, p312]. The point $\mathbf{X}$ is then found using the SVD method by solving the equation $A\mathbf{X} = 0$.

3 Implementation

The algorithms and user interface of the program created were written in the Python language[6]. This enabled rapid development and the use of the NumPy[5] package. NumPy provides matrices, vectors and a number of operations on them. A NumPy array also provides the basis for images, where a 320×240 colour image is represented by a $240 \times 320 \times 3$ array. See Figure 5 for part of the interface.

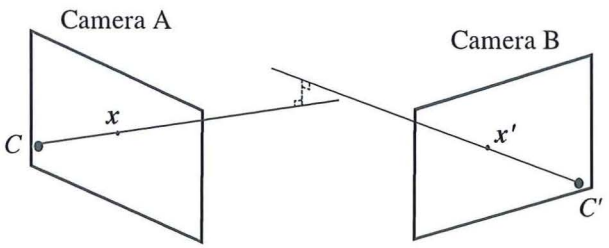


Figure 4: The back projection of corresponding points in the two images where the camera matrices are known. Inaccuracies in camera matrices and imaged point locations cause lines to be skew.

Figure 5: The user interface, showing the image processing pipeline. Clockwise from top left: the original image; the vertical first derivative of the image; the Hough transform space; the Canny edge detection output.

For image processing, the C language was used. This provided a means to access and manipulate individual pixels of an image at high speed. All functions created in C had an interface so that they could be run from and used in the Python code.

In Section 2, the mathematics of reconstructing a scene were described. For this to be done, a number of data inputs must be given.

For the camera calibration, three imaged squares need to be provided along with the location of their corners. This need occur only once for a particular camera, so manual selection of corners is a reasonable assumption.

In the case of the scene reconstruction, the position of corresponding points between two images is all that is required. No assumptions about the shape of objects need to be known. However, this is still a rather difficult task for computers to carry out, as useful information must be extracted from images.

3.1 Image Processing

Image processing is an inherently difficult process when the aim is to extract useful information for later use by a computer. This is mainly because a typical image contains around 80 thousand pixels. Each pixel then contains 3 channels for red, blue and green, which each have a value between 0 and 255.

The end result from image processing is generally to reduce the information to more abstract forms. This can mean locating regions of interest, like points lines and curves which have a mathematical form and are easy to process further. This will give data which is substantially smaller in quantity, but still holds much information about the original image. It is also important that these features are mostly unchanged between consecutive images.

The information in an image can be reduced by 2/3 by converting the image to gray-scale *i.e.* into a single channel. This is very fast and easy to compute, yet removes unnecessary information; objects are no less discernible with only the intensity of light.

Another problem when dealing with images is the presence of noise. When pointing a camera at a static scene and with the camera held still, the change in pixel brightnesses between consecutive images is very apparent. This is often overcome using a smoothing operation such as Gaussian blur. However, this can make edges less defined.

3.2 Canny edge detection

The Canny edge detection algorithm[1] is a useful algorithm for finding edges in an image. The result is that it produces a new image of the same dimensions as the first. However, the pixels are all binary, so they can only take one of two values: black or white. The white pixels then represent edges. Also, the edges are all only one pixel thick which is easier to handle than if they were thicker.

The algorithm works in a number of steps. First the image is smoothed by a Gaussian blur. This means that each pixel takes on a value which is a weighted average of its neighbours. The further away a pixel is from a pixel, x, the less it effects pixel x.

Another filter, called a Sobel operator is then passed over the image. This calculates the first derivative in the horizontal and vertical directions. The horizontal pass uses the kernel

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & -1 \end{bmatrix} \tag{36}$$

This gives the weightings of the surrounding pixels to compute the value of the central pixel. The transpose of this kernel is used for the vertical pass. From the horizontal and vertical passes, two new images are produced.

From the gradient images, two things can be calculated: the magnitude of the gradient at a point, and the direction of maximum gradient change. The magnitude is $V_{mag} = \sqrt{V_H^2 + V_V^2}$, where V_H is the horizontal gradient of pixels, and V_V is the vertical gradient. This can be approximated by $V_{mag} = |V_H| + |V_V|$, which is much easier to compute. The gradient direction is $\theta = \tan^{-1}\left(\frac{V_V}{V_H}\right)$. Again, this is computationally expensive, and only an accuracy of 45° is required.

The next stage is non-maximal suppression. This makes all pixels which are not at the top of a 'hill' go to zero. This is done by comparing a pixel with those in either direction of maximum gradient. If a pixel has greater magnitude than those two in both directions, then it could be an edge. A final check is done to see whether it is above either of two thresholds. If it is, then the pixel will become one of two non-zero values, depending on which threshold it reached.

The final stage is hysteresis thresholding. This is a method whereby the two thresholds are used to determine whether a non-zero pixel is part of an edge. If a pixel is above the higher threshold, then it becomes an edge. A pixel which is only above the lower threshold needs to be adjacent to a true edge pixel (which may have been above either threshold) in order for it to become an edge.

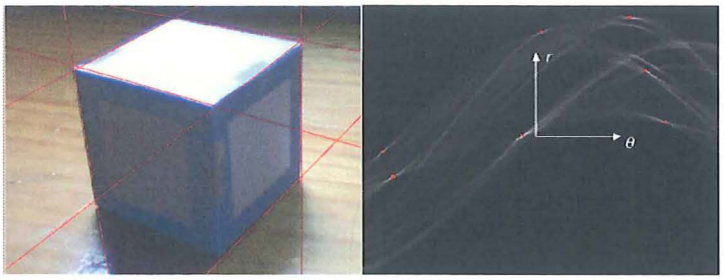
The Canny edge detection algorithm was implemented by the author in C. The resulting binary image was used in the Hough transform.

3.3 Hough Transform

The Hough transform is an algorithm used to find straight lines within an image by a voting procedure.

All straight lines in 2D can be defined by two parameters. In the Hough transform, a line is defined by its direction, and its perpendicular distance from the origin.

A pixel which is part of an edge in the Canny edge detector image could be part of a line in any direction away from that point. That point then votes for all those possible directions. The votes are done in a



(a) Original image with lines found (b) Hough parameter space

Figure 6: The result of a simple Hough transform of an image. The transform is taken of the binary image generated by the Canny edge detector. The largest maxima in the Hough parameter space are located, and the lines are reproduced on the original image.

parameter space, *i.e.* another image. In this parameter space, the horizontal axis is the line angle, θ, and the vertical axis is the distance, r, from the origin.

When a number of points lie in a line, that line, represented by a point in the parameter space, will receive more votes than other candidate lines near it. Thus, local maxima in the parameter space indicate the presence of a line. A Hough transform like that described is shown in Figure 6.

In this simple form, the Hough transform has a number of problems. It is difficult to find the local maxima. This is because using a simple threshold will locate multiple lines in a single location, and will fail to identify, for example, lines which are too short. The quantity of edge points in an image will raise the total number of votes, while none of those may represent a true line. This last problem is apparent in the Hough transform in Figure 5.

Another problem is that the transform does not provide any information about the start and end points of lines. This information is required in order for corners to be detected.

The algorithm can also be quite slow with a large number of edge points.

Some of these problems were overcome by using a faster Hough transform algorithm as described in [3]. The algorithm, written by [3], was incorporated into the author's program. Due to the nature of the algorithm, start and end points of lines could be determined approximately, and used to find corners. However, this was still not robust, and suffered from corner points changing apparently randomly. It was thus difficult to use it for tracking feature points.

3.4 Point Tracking

Given the difficulties with the Hough transform method, a new algorithm was found in OpenCV[2]. This worked by calculating "optical flow" between images at key points. Using this, it was possible to track points between frames. Two frames could then be selected by the user, in which correspondences were known. However, this still suffered from a certain amount of drift, where some points may move relative to the point they are tracking.

4 Results

The camera calibration was successful in generating an appropriate camera matrix. Figure 7 shows three views, each with a hand-selected square. The points of these squares were used in the algorithm created from Section 2.1. This resulted in the calibration matrix

$$K = \begin{bmatrix} 666 & 6.09 & 142 \\ 0 & 668 & 141 \\ 0 & 0 & 1.00 \end{bmatrix} \tag{37}$$

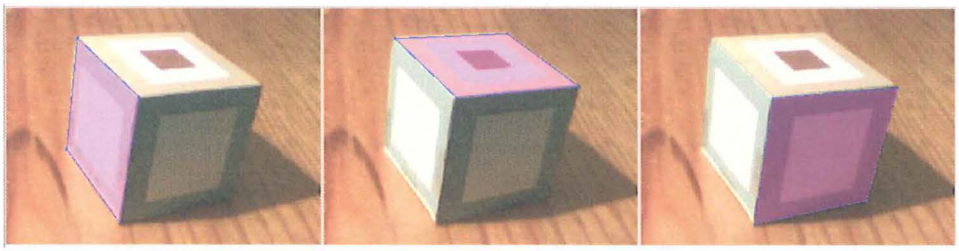


Figure 7: Calibrating the camera using three squares. Note that in the general case, it is not necessary for the three planes to be orthogonal.

Since all further pictures were taken with the same camera, this calibration matrix was used in calculations involving those pictures.

A complete reconstruction is shown in Figure 8. As can be seen, planes which are orthogonal in the true scene also appear to be approximately orthogonal in the reconstruction. This means that the algorithms created work correctly. The errors that are apparent in the views will be mainly due to incorrectly placed points, and errors in the calibration matrix found above.

Linking the feature point detection and tracking with the reconstruction algorithms was largely unsuccessful. This was mainly due to variability and uncertainty in the points generated from the image processing. It was found that the reconstruction was highly susceptible to error in the presence of incorrectly positioned points.

5 Conclusion

This project was successful in generating a 3D reconstruction of a scene, provided that all points are selected manually. The algorithms presented required three squares to be selected from images created by the camera used in order to calibrate that camera. Following that, it was possible to generate a reconstruction where corresponding points are selected in two images of a scene.

Although the examples presented are somewhat contrived, this examination of 3D reconstruction can be extended to reproduce arbitrary scenes in a more automated fashion. More work needs to be done in order to make the reconstruction algorithms more robust to noise factors. Another possible improvement is to refine the point tracking algorithm used in Section 3.4 to remove points that track erroneously.

References

[1] J Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, 1986.

[2] Intel Corporation. OpenCV: Open source computer vision library. `http://opencv.willowgarage.com/wiki/Welcome`, 1999–.

[3] Leandro A.F. Fernandes and Manuel M. Oliveira. Real-time line detection through an improved hough transform voting scheme. *Pattern Recognition*, 41(1):299 – 314, 2008.

[4] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, Cambridge, 2003.

[5] Travis Oliphant et al. NumPy: Open source scientific tools for Python. `http://numpy.scipy.org/`, 2001–.

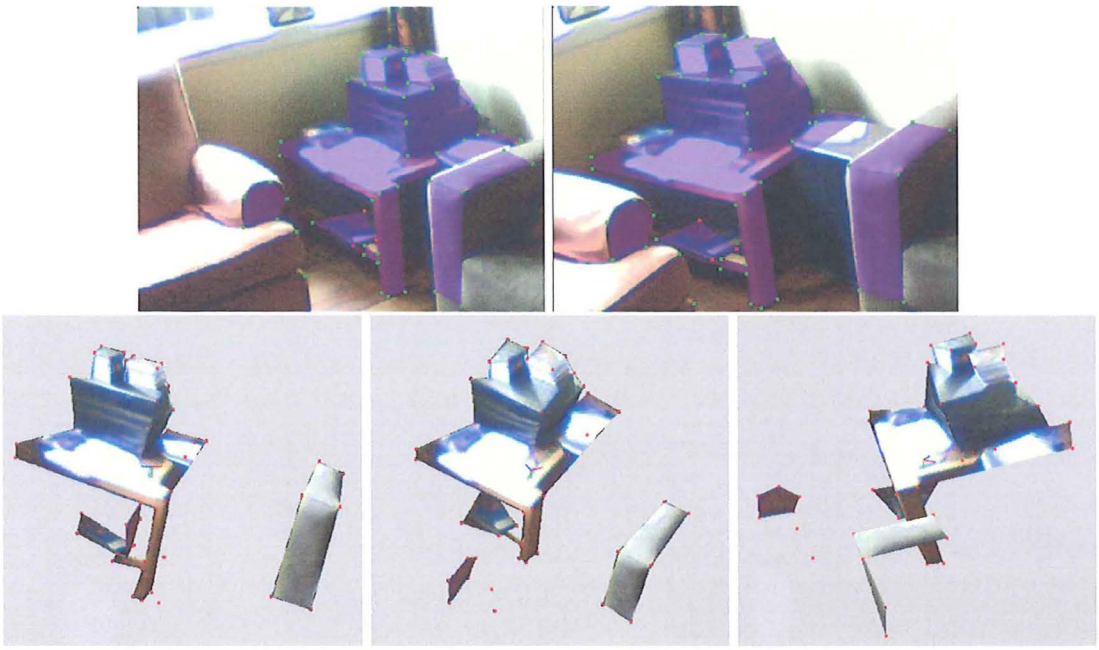


Figure 8: The reconstruction of a scene. The points are selected by hand in the two images. After calculation, the reconstruction is formed, and can be viewed from an arbitrary angle.

[6] Guido van Rossum et al. Python: Open source interpreted, interactive, object-oriented programming language. `http://www.python.org/`, 1991–.

[7] Eric W. Weisstein. Cholesky Decomposition from MathWorld – a Wolfram web resource. `http://mathworld.wolfram.com/CholeskyDecomposition.html`.

[8] Eric W. Weisstein. Singular Value Decomposition from MathWorld – a Wolfram web resource. `http://mathworld.wolfram.com/SingularValueDecomposition.html`.